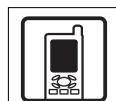
続 実設計に応用できる演算回路スキルを身につけよう

システムの記事

外村元伸

第6回 除算・開平・逆数・逆開平数とその演算回路設計(3)逐次演算を加速する方法

商，開平，逆数，逆開平ディジットを１ビットずつ逐次的に決定していく方法は，けた数に比例した演算時間がかかる．特に，部分剰余演算よりもディジット選択のための論理回路の段数が多く必要である．そこで，逐次決定法をベースにしながらも，演算を少しでも高速化する方法がいくつか提案されている．今回は，これらのアイデアのいくつかを紹介する．（筆者）

● ディジット選択論理と部分剰余演算の制御

除算，開平，逆数，逆開平演算が共通に設計できる基数2の逐次アルゴリズムに関して，部分剰余演算結果の上位3ビットのキャリ伝播と4ビット目のキャリ・セーブ値を参照することで，1ビットずつ各ディジットを決定できる選択規則（**表1**）を前回（2007年7月号，pp.126-132）導きました．

ディジット選択規則を導く過程は大変でしたが，規則そのものは単純です．ディジット選択論理とそれによって制御される部分剰余演算（iけた目）の回路を**図1**に示します．

表1　ディジット選択規則

q_i	r_{-1}	r_0	r_1	$(s_2,\ c_2)$
+1	0	1	1	＊
+1	0	1	0	＊
+1	0	0	1	＊
+1	0	0	0	Not 0
0	0	0	0	0
0	1	1	1	＊
−1	1	1	0	＊
−1	1	0	1	＊
未定義	1	0	0	＊

部分剰余演算結果の上位3ビットのキャリ伝播（$r_{-1}r_0r_1$）の符号（r_{-1}）によって，$r_{-1}=0$のときは$q_i=+1$（X反転＋1加算），$r_{-1}=1$のときは$q_i=-1$（X加算），そして4ビット目のキャリ・セーブ値（s_2，c_2）を参照して，（$r_{-1}r_0$r$_1$，（s_2，c_2））＝（000，（0，0））または（$r_{-1}r_0r_1$，（s_2，c_2））＝（111，＊）（＊：don't care）のときだけ，$q_i=0$（$X=0$加算）となるように制御して部分剰余に加算する論理を組みます．これにはセレクタ論理を巧みに利用しています．Xを反転したときには，最下位けたに＋1を加えることを**図1**には書いていないので，実際の設計においては忘れないように気をつけ

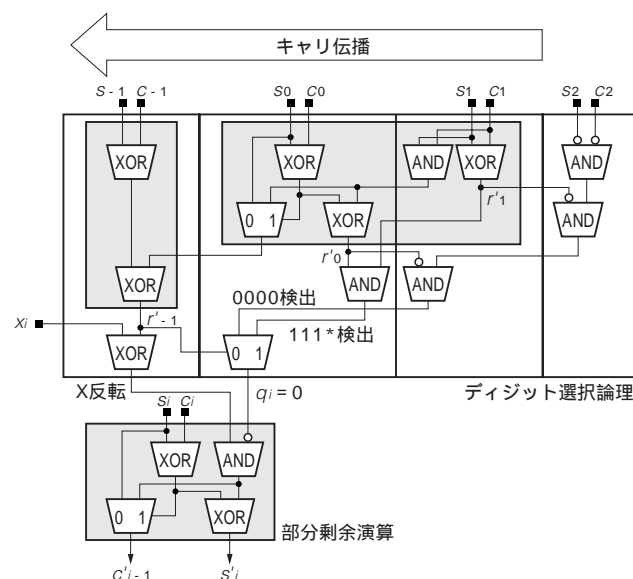
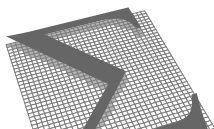



図1　ディジット選択論理と部分剰余演算の制御
上位3けたのキャリ伝播と4けた目のキャリ・セーブ値を参照してディジット選択論理を組み，部分剰余演算回路を制御．

KeyWord　ディジット選択論理，部分剰余演算，除算，開平，逆数，逆開平演算，キャリ伝播，プリスケーリング法

てください．

さて，**図1**を見て分かるように，ディジット選択論理の方が，部分剰余演算よりも論理段数が2倍あります．そのため，ディジット選択論理の部分をなんとかしたくなります．また，逐次アルゴリズムのため演算長に比例した時間がかかることから，1ディジットを決定し部分剰余演算する1サイクル当たりの論理段数を実質的に減らしたいという要望があります．そこで，今回は論理段数を減らし逐次演算を加速する以下のいくつかのアイデアについて解説します．

①ディジット選択候補すべての部分剰余演算を並列実行し，ディジット決定結果で部分剰余演算結果を選択する方法
②除数の範囲を限定し，ディジット選択規則を簡単化する方法：プリスケーリング法
③一度にm(≧2)ビットずつディジットを決定していく方法：高基数法

図3　ディジット選択論理の2段（多段）重ね接続

1クロックの間に実行できる総論理段数が許す限り，ディジット選択論理を多段に重ねる．

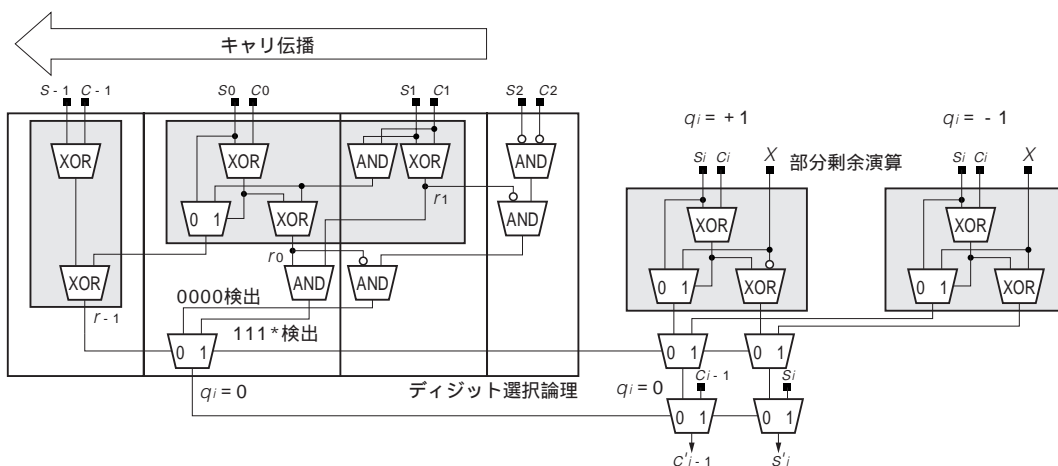
● すべての部分剰余演算を並列実行する

演算回路の論理段数を減らしたいということで，すぐ思いつくのが，ディジット決定結果で部分剰余演算結果を選択する方法です．**図2**に示すように，ディジット選択論理による結果を待たずに，q_i＝＋1，0，−1すべての場合の部分剰余演算を並列に行います．ただし，**図2**のように部分剰余演算が2段でできる回路では，選択のためにセレクタ1段分遅れます．**図1**の方式に比べて1段分速くなるだけです．1段分でも速くしなければならないときには有効な方法です．それは，**図3**に示すように，ディジット選択論理を一度に2段（多段）重ねて接続して，実質的に1クロック（マシン・サイクル）あたり2ビット（多ビット）のディジット決定を可能にするような場合に有効です．1クロックあたり論理n段が可能な場合，**図2**の5段の回路が，n/5個分重ねて接続できる計算になります．

● キャリ伝播も含めた並列部分剰余演算結果を選択する

q_i＝＋1，0，−1すべての場合の並列部分剰余演算をキャリ伝播も含めて行うことも考えられます．その場合を**図4**に示します．上位3けたのキャリ伝播後の現在の部分

図2
並列部分剰余演算結果をディジット選択論理結果で選択

ディジット選択論理と並列にq_i＝＋1，0，−1すべての場合の部分剰余演算をあらかじめ実行し，ディジット決定値によって部分剰余演算結果を選択．

剰余演算結果($r_{-1}r_0r_1$)の符号(r_{-1})だけによって，$q_i=+1$($r_{-1}=0$でX反転＋1加算)または$q_i=-1$($r_{-1}=1$でX加算)が決定します．r_{-1}とXの排他的論理(XOR)を行い，次の部分剰余演算とそのキャリ伝播を一気に実行します．そのあいだに$q_i=0$($X=0$加算)，つまり($r_{-1}r_0r_1$，(s_2，c_2))＝(000，(0，0))または($r_{-1}r_0r_1$，(s_2，c_2))＝(111，∗)(∗：don't care)であるかを判定し，最終的にセレクタで次の部分剰余演算結果を選びます．

ここで，部分剰余の初期値では，非キャリ・セーブ形式($r_{-1}r_0r_1$)になっていることに気をつけてください．部分剰余演算結果は左1ビット・シフトするため，$q_i=0$($X=0$加算)の場合でも上位4けた目のキャリ・セーブ形式を非キャリ・セーブ形式に変換する必要があることから，**図5**に示す回路によって，けた上げのインクリメント処理を行います．

r_0とr_1に対してけた上げのインクリメントを行う回路には工夫したものを採用しています．一見すると，3-2カウンタ回路に見えますが，r_0とr_1はけたが異なり，同じけた同士のカウンタ回路ではありません．下位からのけた上げ入力が来たときに，＋1のインクリメント処理を行う回路として働いてくれます．$r_0=r_1=1$のときに下位からけた上げ入力が来ると$r_0=r_1=0$にクリアされます．けた上げ入力がないと$r_0=1$がセレクトされ，r_0はクリアされません．

さて，これだけがんばっても1ビットのディジット値を求める一連の処理において，論理段数が6段かかっており，**図1**に示した方法も同じく6段です．これでは効果がないように見えますが，最終けたのディジット値のキャリ伝播分だけ削減しています．**図2**の方法は面積が大きくなるが5段なので，どの方法を選ぶかは，速度と面積の兼ね合いになります．

● 除数の範囲を限定しディジット選択規則を簡単化する

除算Y/Xは，$1/2\leqq X$，$Y<1$の範囲に正規化していますが，もっと扱いやすい範囲はないのでしょうか．除算の次の性質に注目してください．

$$\frac{Y}{X}=\frac{MY}{MX}=\frac{Y'}{X'} \quad\cdots\cdots(1)$$

被除数Yと除数Xの両方をM倍しても除算値は変わりません．この性質を利用して，除数Xの範囲を適当に制限して除算のディジット選択規則が簡単化できることに気づい

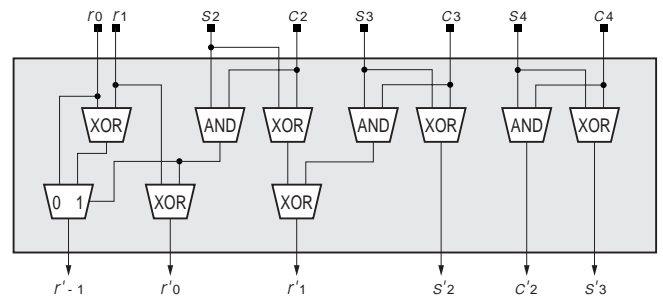



図5　$q_i=0$($X=0$)の場合のけた上げのインクリメント回路
r_0とr_1に対して，けた上げのインクリメントを行う回路の特徴に注意．

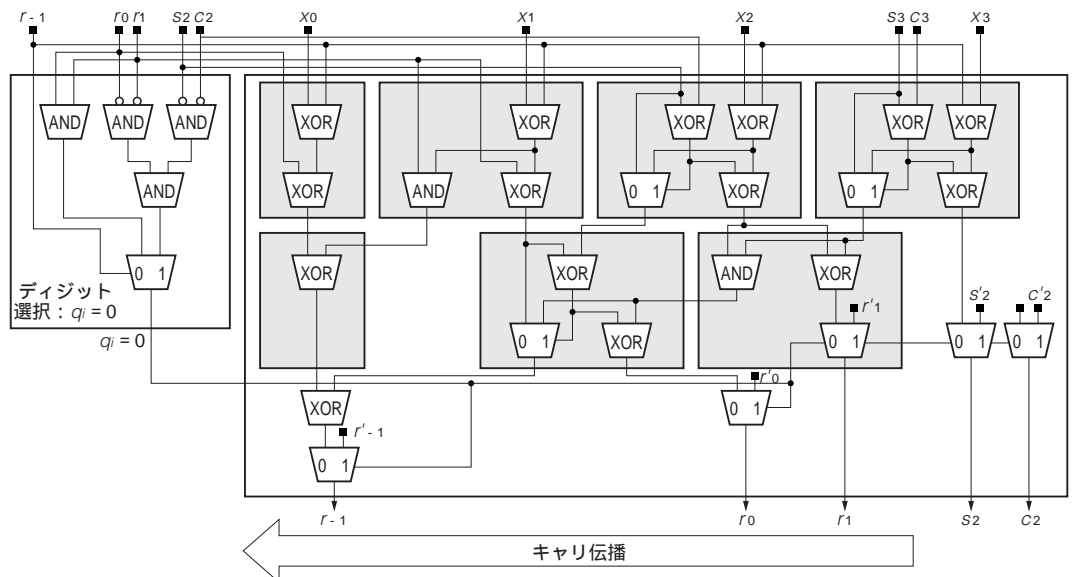



図4　キャリ伝播も含めた並列部分剰余演算結果を選択する回路
符号ビットr_{-1}によって，現在の部分剰余演算値にXまたはXの反転＋1を加算しキャリ伝播を実行し，その間に$q_i=0$の場合かを判定し，次の部分剰余演算結果を選択．

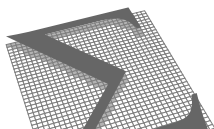



表2　除数のプリスケーリングによるディジット選択規則とその簡単化（*＝{2, 1}）

q_i=+1	加数	q_i=−1	加数	q_i=0	加数	q_i=0	加数
	101		010		111		000
被加数 01	次部分剰余 1*	被加数 10	次部分剰余 1*	被加数 00	次部分剰余 1*	被加数 11	次部分剰余 1*
	11 00		11 00		11 00		11 00

図6　プリスケーリング後のディジット選択論理と部分剰余演算の制御

プリスケーリングによってディジット選択論理を簡単化し，少ない論理段数で実現．

たのが，チェコのSvoboda［1963年］(3)やインドのKrishnamurthy［1970年］(2)でした．後にErcegovac(1)がプリスケーリング法と呼んで詳しく研究しました．筆者もディジット選択規則の簡単化を試みていたら，適当な範囲があることに気づきました．そこで，彼らの論文があることを知ったのです．開平の場合は，プリスケーリングすると$\sqrt{X'}=\sqrt{MX}$から，$\sqrt{X}=\sqrt{X'}/\sqrt{M}=\sqrt{M'}\sqrt{X'}$となるので，後にも乗算が必要になります．

表2に示すように，除数Xを$1/2 \leqq X < 3/4$の範囲に，つまり$[0.10\cdots]$b＝[1/2, 3/4)に限定した場合，商ディジットは部分剰余の上位2けただけの参照で決定できることを示しています．上位2けたは，$(r_0,\ r_1)$とし，r_0のけたを符号部とします．$X=[0.10\cdots]$bであることから，部分剰余の上位2ビットが[0.1]bのとき，商ディジット$q_i=+1$で$-X=[0.10\cdots]$b＋$[0\cdots1]$bを加算，[1.0]bのとき$q_i=-1$で$X=[0.10\cdots]$bを加算，[0.0]bのとき，$q_i=0$で$0=[1.11\cdots1]$b＋$[0\cdots1]$bを加算，[1.1]bのとき$q_i=0$で$0=[0.000\cdots0]$bを加算します．ここで，[0.0]bのときに，$q_i=0$で安易に$0=[0.000\cdots0]$bを加算しないのは，最上位ビットを捨てて左1ビット・シフトすると符号の連続性が保たれない場合，例えば[0.02]bのとき，$[0.000\cdots0]$b加算により，結果[0.1]bとなる場合があるからです．そうすると，いずれの場合もキャリ伝播後の部分剰余演算結果は，[1.1]bまたは[0.0]bとなることから，最上位ビットを捨てて左1ビット・シフトしても符号の連続性が保たれます．このように$q_i=0$の場合，最上位ビットの符号を見て，$0=[1.11\cdots1]$b＋$[0\cdots1]$bか$[0.000\cdots0]$bを加算するようにします．論理反転の制御もしやすくなります．**図6**にプリスケーリング$Q=Y'/X'=(MY)/(MX)$を行った除数X'と被除数Y'を用いて，上位2けただけをキャリ伝播させてディジット選択し，部分剰余演算を行う回路を示します．

次にプリスケーリング$Q=Y'/X'=(MY)/(MX)$のためのM倍変換について説明します．除数Xを$[0.10\cdots]$b＝[1/2, 3/4)の範囲に収める必要があります．この場合そんなに難しくありません．正規化されたXは，$[0.1\cdots]$b＝[1/2, 1)の範囲にあるので，**表3**に示すように，$[0.x_1x_2]$b＝[0.10]bのときは，明らかに$M=[m_0.m_1m_2]$b＝[1.00]bです．また，$[0.x_1x_2]$b＝[0.11]bのときは，$[m_0.m_1m_2]$b＝[0.11]bとした場合，$X'=MX=[0.10\cdots]$bとなることが分かります．それで，x_2ビットを見て，m_0，m_1，m_2のビットの値を決めればよいので非常に簡単です．つまり，

$$[m_0.m_1m_2]\text{b}=[\bar{x}_2.x_2x_2]\text{b} \quad\cdots\cdots(2)$$

表3　プリスケーリング変換の倍数M

X：$0.x_1x_2$	M：$m_0.m_1m_2$
0.10	1.00
0.11	0.11

です．プリスケーリング変換の回路を**図7**に示します．式(1)から除数Xと被除数Yを同じようにM倍します．M倍のためのセレクタ回路は**図7**の左に示す通りです．M倍の結果はけた上げ先見加算回路（CLA）で求め，それぞれX'，Y'を得ます．プリスケーリング方式は，その変換回路が余分になっていますが，実際の除算器では，ゼロ除算をしないように除数Xのゼロ検出を行います．これがCLAと同じようにトーナメント形式（けた数nのとき$\log_2 n$オーダ）の

回路になることから，ゼロ検出の回路と並列に置けばこのオーバヘッドの影響はほとんどありません．プリスケーリング変換後は部分剰余の上位2けたのキャリ伝播でより簡単なディジット選択ができるため，より少ない論理段数になるという利点が生かせます．

これまで1ビットずつディジットを決定していく基数2のアルゴリズムをベースに扱ってきました．次回は，一度にm（$\geqq 2$）ビットずつディジットを決定していく高基数$p = 2^m$のアルゴリズムを解説します．高基数を扱うことは，たとえ基数4の場合でも格段に難しくなります．

1994年に発覚した，米国Intel社のPentiumプロセッサの浮動小数点除算バグ[(4)]は当時話題になりました．従来，基数2で実現していたのを，高速化のために基数4のアルゴリズムを採用したのですが，ディジット選択を行うために作成されたルックアップ・テーブルにおいて，あるケースのエントリが欠落していました．

● ディジット選択規則（表1）の未定義部についての補足

ディジット選択規則（**表1**）において，$(r_{-1}, r_0, r_1, (s_2, c_2)) =$ [10.0＊] bの場合は，q_iの値を未定義としているのはなぜかということに関する説明が不足していましたので補足します．

もし$q_i = -1$と定義すると，例えば，除数$X =$ [00.11] bの例では，部分剰余演算結果が，[10.00] b＋[00.11] b＝[10.11] bとなることから，[10.11] bを左1ビット・シフトして最上位ビットを捨てる（[101.1] b → [01.1] b）と，符号が1から0になり符号の連続性が保てません．そこで，$(r_{-1}, r_0, r_1, (s_2, c_2)) =$ [100＊] bとなることを禁止するため，未定義にします．

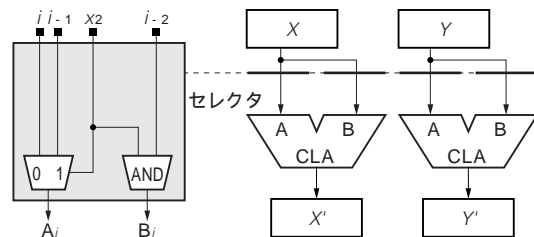



図7 プリスケーリング変換回路
除数のx_2ビットを参照して倍数Mのセレクト制御し，けた上げ先見加算器（CLA）にてプリスケーリング変換を実現．

参考・引用＊文献

(1) M. D. Ercegovac and T. Lang ; Division and Square Root, Digit-Recurrence Algorithms and Implementations, Kluwer Academic Publishers, 1994.
(2) E. V. Krishnamurthy ; On Range-Transformation Techniques for Division, IEEE Transaction on Computers, Vol.C-19, No.2, pp.157-160, Feb. 1970.
(3) A. Svoboda ; An algorithm for division, Information Processing Machines, Vol.9, pp.25-34, 1963.
(4) T. Coe ; Computational Aspects of the Pentium Affair, IEEE Computational Science & Engineering, Vol.2, No.1, pp.18-31, Spring 1995.

とのむら・もとのぶ
大日本印刷株式会社　電子モジュールセンター　DNPひびきの研究所

<筆者プロフィール>
外村元伸．除算・開平・逆数・逆開平数の演算器設計は奥が深く，かつて筆者の研究のメイン・テーマの一つでした．忘れてしまわないうちに，あまり知られていないことがらを含めて，まとめておこうと思って紹介しています．